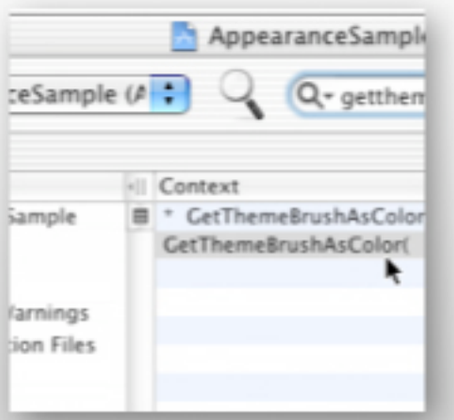
Debugging Services for Mac OS X

Session 308



Debugging Services for Mac OS X

Jeff Glasson
Manager, Debuggers/Runtimes

What You'll Learn

- Getting started debugging
- Neat new debugging features in Xcode
- Advanced gdb command line debugging
- Other helpful tools
- Lots of demos/examples

Debugging 101

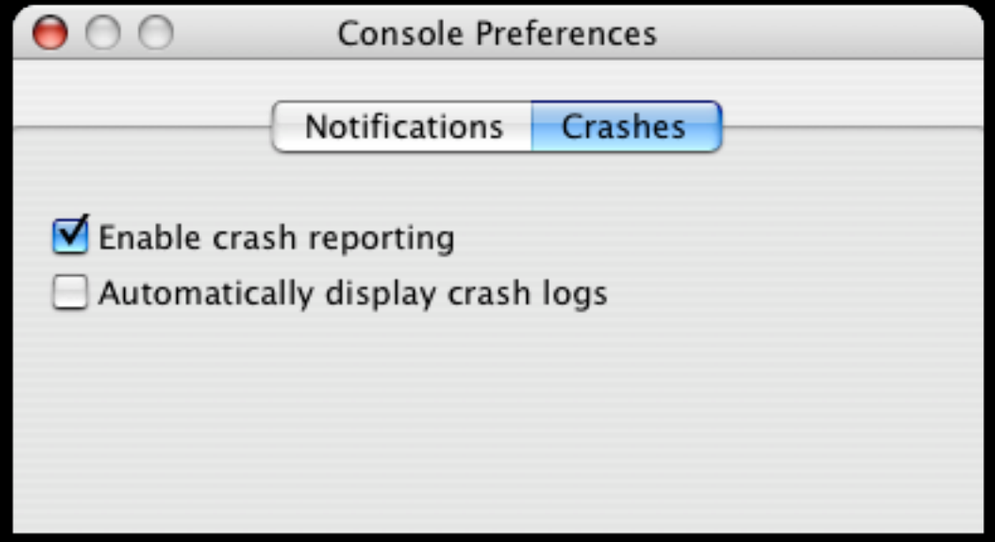
- Why did my app crash?
- Why did my app hang?
- What's that strange behavior…
  and how can I fix it?

Getting Started

- The arsenal of UI tools:
  - Sampler
  - MallocDebug
  - CHUD
  - gdb via Xcode

How to Get a Crash Dump

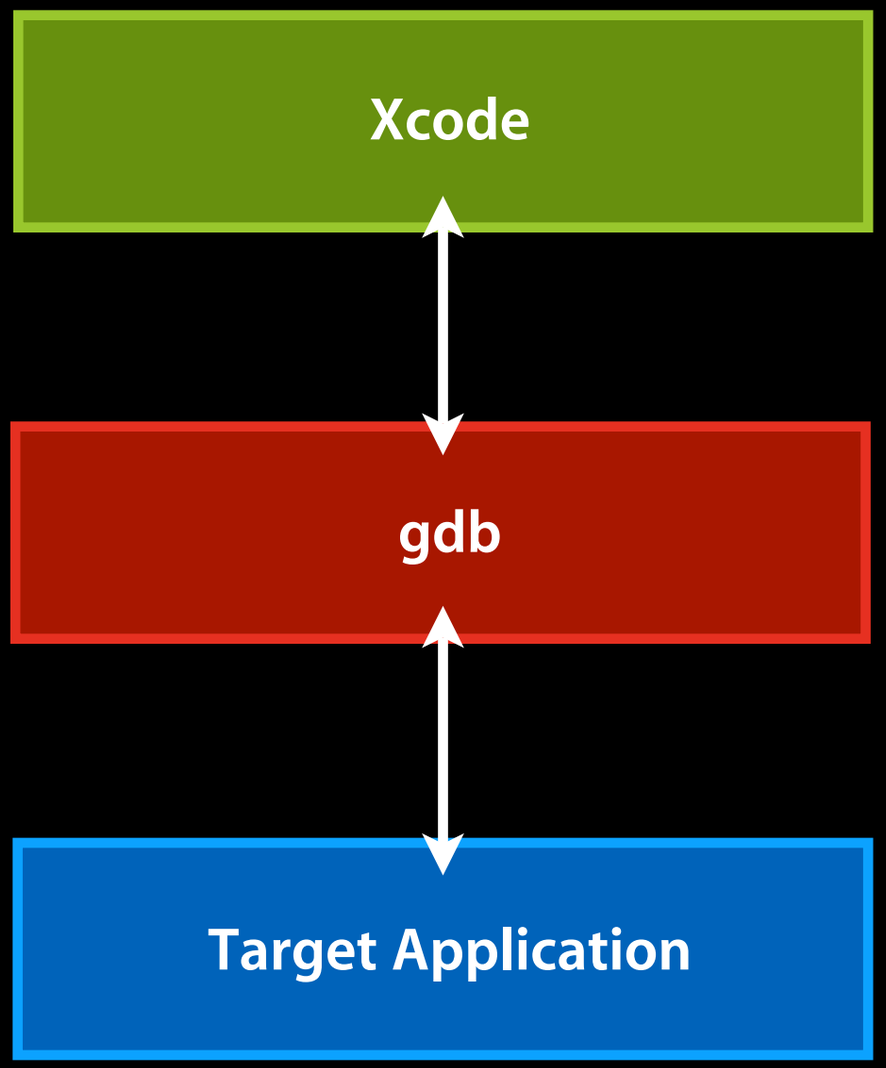
- Turn on crash logging on Console:
  /Applications/Utilities/Console



Crash Logs In Console

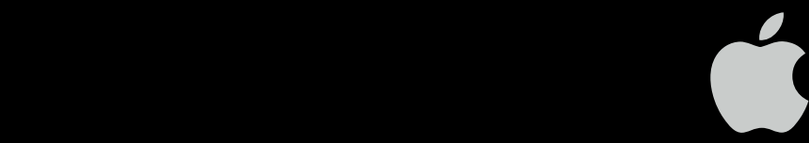
- Logs appear in:
  ~/Library/Logs/CrashReporter/<app name>.crash.log
  - Latest crash at the end of log
  - *Warning!* Contents saved across reboots
- Logs include:
  - Stack trace for each thread
  - Register state for thread that crashed
  - File and line number if binary has debug symbols

Mac OS X Debugging Architecture



Getting Started Debugging In Xcode

- GUI interface to gdb
  - View threads, stacks, local variables, expressions
  - Set/remove breakpoints, control program execution
  - Only method to access cool new features
- Xcode also supports debugging non-C based languages
  - AppleScript
  - Java
  - Not the topic of this talk



Configuring for Debugging

- New or upgraded projects have correct settings
  - Development Build style
- Older or legacy projects
  - Generation of symbolic information enabled (-g)
  - Code optimization set to lowest level (-O0)

Framework Debugging

- Make sure to set the framework path to debug version:

Major New Debugging Features

- Custom data formatting
- Fix and Continue
- Support for C++ exceptions
- New debugging UI

Custom Data Formatters

- Summarize data
- Custom plugins
- Custom text with data values
- Evaluate expressions

Custom Data Formatters

- Bound to a type
- Changing one changes all
- 35 Carbon formatters (in the Xcode preview)
- 35 Foundation formattters (in the Xcode preview)

Demo

Xcode Debugger UI

Scott Tooker
Xcode Engineer

Fix and Continue

- Debug without stopping
  - Make changes to a running program
  - Continue debug session without recompile/link

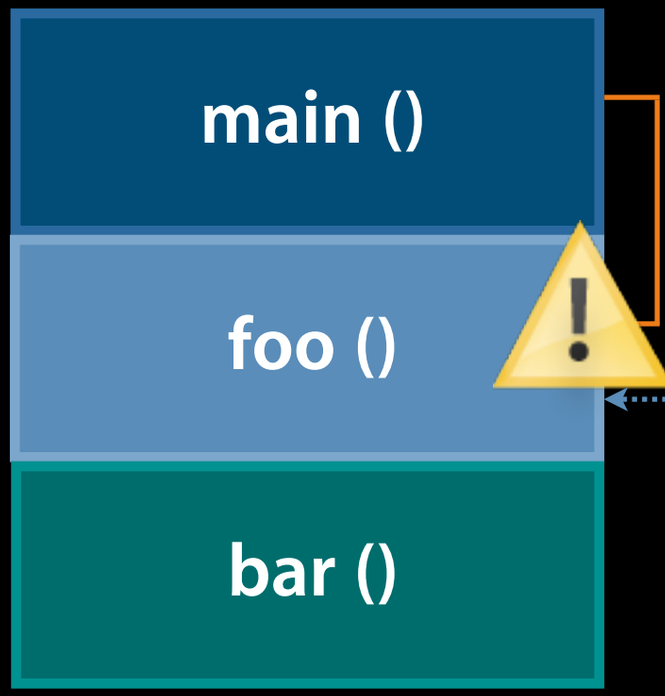
Fix and Continue Highlights

- Depends on tight integration between Xcode and gdb
- Not all types of changes allowed
- Only supported with gcc 3.3
- Works with Zero Link too

Why Should I Care?

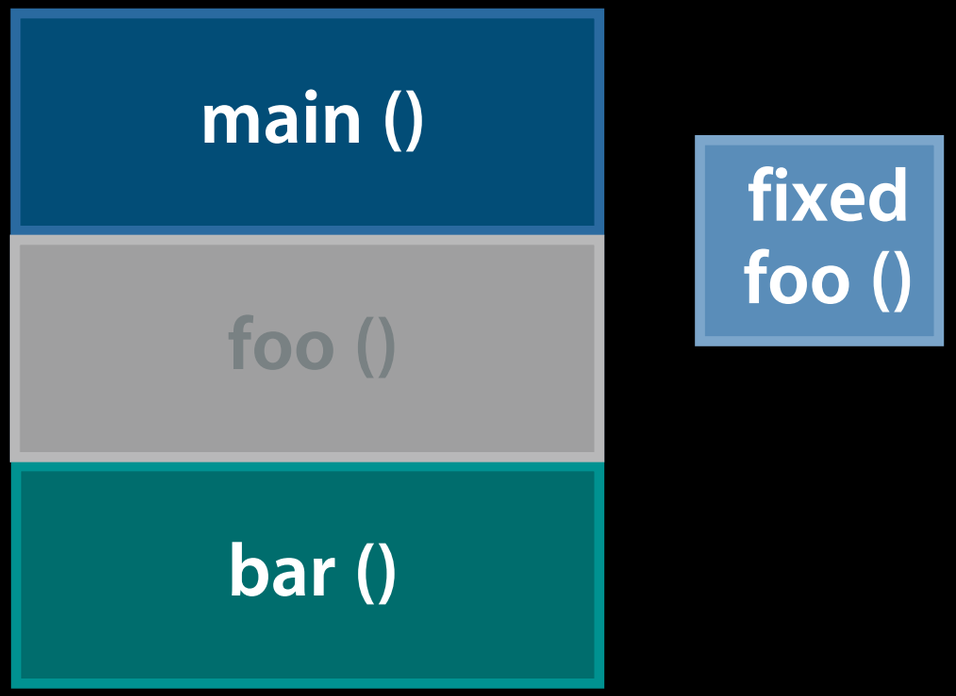
- Extremely useful to continue long debug sessions
- Allows prototype-style development

How Fix and Continue Works

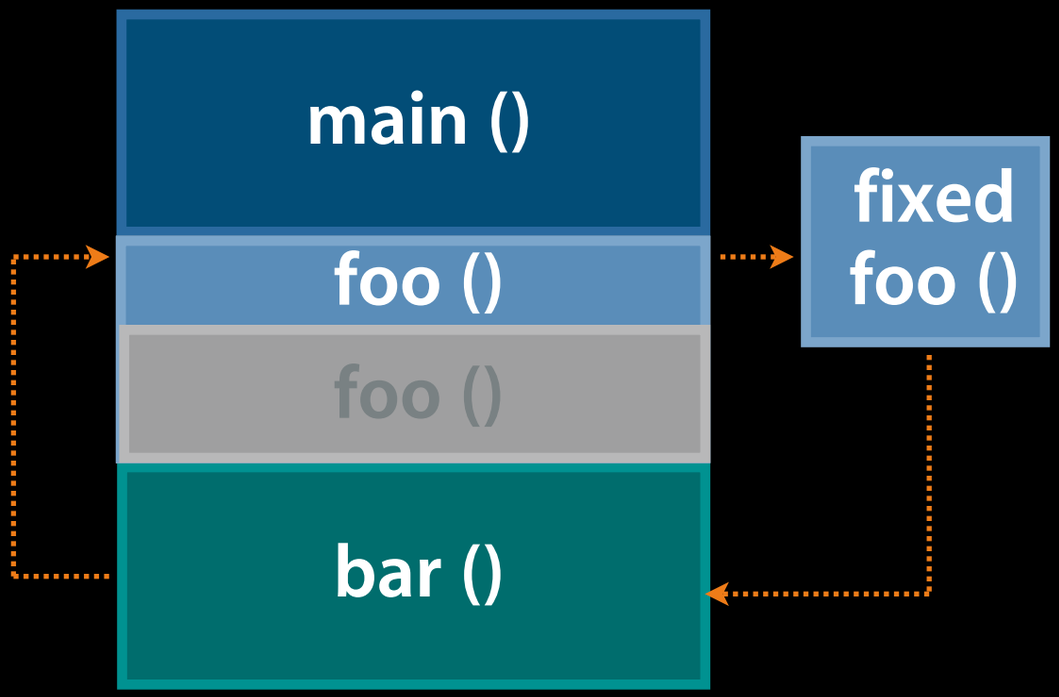


1. Edit
2. Compile
3. Xcode tells gdb to load fixed foo ()

How Fix and Continue Works



How Fix and Continue Works



Limitations

- Doesn’t support all types of changes
- Common “unfixable” changes will be flagged in Xcode
- Is not supported for use via the gdb command line

List of Changes Known to Not Work

- Changing structure/class layouts
- Change function signature
- Adding global or static data
- Adding anything that needs runtime initialization
- Adding virtual member functions
- If a changed function is on top of stack:
  - Adding local data
  - Adding objects initialized earlier in function

So, What Does Work?

- Logic errors
- Adding new calls to existing routines
- Deleting code from existing routines
- Changing constants
- Other common operations…

Fix and Continue Tips

- Remember:
  If Fix and Continue fails, re-run from the beginning
- *Note:* Always rebuild after Fixing
  - Makes sure the changes are incorporated

Demo

Fix and Continue

Scott Tooker

C++ Exception Support

- Break on catch and/or throw
- Simple UI to control
- Command line allows more sophisticated usage

Caveats

- Lots of custom formatters may slow debugging
- Don’t call code that takes a lock
- Large Zero Link applications may slow debugging

Advanced gdb Features

Klee Dienes
Senior Debugger Engineer

Cached Symfiles

- New gdb performance improvement
- Enables caching debug information
  - Improves debugger startup time
- Transparent for the user
- Installation creates caches for system frameworks
- Optionally create caches for user binaries

Possible Cached-Symfile Issues

- Cached symfiles for system libraries may need to be refreshed after an update
  - warning: Pre-compiled symbol file "libSystem.B.dylib.syms" is out of date
- Can be removed and regenerated at any time
  - sudo rm -r /usr/libexec/gdb/symfiles
  - sudo /usr/libexec/gdb/cache-symfiles

Cached Symfiles for User Binaries

- Enable in .gdbinit
  - set generate-precompiled-symfiles 1
  - Can work on a per-project basis
- Excellent for collections of libraries, Zero Link binaries, large binaries that change infrequently
- Inappropriate for large binaries that change often
- Cache will be stored in ./gdb-symfile-cache
  - To disable, remove the symfile cache

Watchpoints

- Stops whenever the watchpoint expression changes
  - Useful for monitoring data values
  - Any expression is valid as the watch condition
- Implemented by trapping memory page accesses
  - High overhead when data is frequently modified
  - Don’t use watchpoints on local stack variables
- Examples
  - watch {char[1024]} buf
  - watch head->next->next

Breakpoint Conditions/Commands

- Any valid expression can be used as a condition
  - Conditions work on watchpoints, too

```
watch globalnum
cond (globalnum == 10)
```

- Commands applicable to any breakpoint or watchpoint

```
break open
commands
        printf “Opening file \”%s\”\n”, (char *) $r3
        continue
end
```

Using MallocDebug

- Encourages badly behaved programs to crash
  - Overwrites freed memory
  - Adds guard words before and after blocks
- Writes warning messages to Console
- Can be used with gdb
  - set env DYLD_INSERT_LIBRARIES /usr/lib/libMallocDebug.A.dylib
  - set env DYLD_FORCE_FLAT_NAMESPACE 1
  - rbreak malloc_printf
- Documentation: use the Help menu

Attach

- Attach to any already-running program
  - attach <name/pid>
  - Supports name/pid completion
- Useful for programs that may be complicated to start
  - Or for working with Sampler/MallocDebug
  - sleep(600) call in program can make it easier to attach

Sample

- Command-line version of Sampler
  - Works with any binary
    - E.g., sample <pid> <duration>
  - Output written to /tmp/<app>_<pid>.sample.txt
- Attach to running programs
  - gdb in the Terminal
  - From Xcode: use console window

Wrap Up

Summary

- Greatly improved debugging experience
- New features improve performance / productivity
  - Fix and Continue
  - Custom Data Formatters
  - Cached symfiles

How to Learn More

- Part of Xcode tools package
  - Use the Documentation window and search features
  - man pages for gdb
- Inside Mac OS X books
  “Inside Mac OS X: System Overview”
- Apple Developer Connection tools pages
  http://developer.apple.com/tools/

Who to Contact

Godfrey DiGiorgi
Technology Manager, Development Tools
ramarren@apple.com

Development Tools Engineering Feedback
xcode-feedback@group.apple.com

To Post Bug Reports
http://developer.apple.com/bugreporter

Mailing Lists at Apple
http://lists.apple.com

http://developer.apple.com/wwdc2003/urls.html

Roadmap

310 Debugging and Tuning Carbon Apps
Debugging and performance for Carbon apps

The Presidio
Fri., 9:00am

311 Software Testing Tools
Automate your application testing

Russian Hill
Fri., 10:30am

Q&A

Godfrey DiGiorgi

Technology Manager, Development Tools

ramarren@apple.com

http://developer.apple.com/wwdc2003/urls.html

WWDC 2003

Worldwide Developers Conference